

NEW ALGORITHMS FOR POLYNOMIAL MULTIPLICATION*

DOROTHEA A. KLIP†

Abstract. Exploiting the structure of the 2-dimensional sorting problem associated with the polynomial product has been the strategy in the design of certain algorithms which are faster for a large class of problems than those found in the literature. First a parallel is drawn between GEN-MULT and Horowitz's SORT-MULT algorithm [A *sorting algorithm for polynomial multiplication*, J. Assoc. Comput. Mach., 22 (1975), pp. 450-462]. The former owes its better performance to the global incorporation of the structure. A geometric picture of the relative location of the product exponents enabled us to present a new approach, implemented by the SLICE algorithms, in which the product exponents are sorted in separate parallel slices. For a class of problems of moderate size and sparsity, which was investigated by Johnson [*Sparse polynomial arithmetic*, Proc. Eurosam Conf., SIGSAM Bull., 8 (1974), pp. 63-71] with the ALTRAN system, the simple SLICE-S algorithm gave a 1:4 improvement, timewise and relative to sorting effort, as compared with ALTRAN's LI algorithm. The optimized version, for large, sparse problems, is based on theoretical arguments. It performs in linear time as an average with respect to its major operations.

Key words. polynomial multiplication, systems for symbolic manipulation, time complexity of $X + Y$ sorting

1. Introduction. The importance of efficient algorithms in the field of symbolic mathematics arises from the large storage requirements for mathematical expressions in general combined with the expense of manipulating them. In the case of the polynomial product one has to generate an expression whose size is the product of the size of each of the operands. It is the general consensus that symbolic entities can only be dealt with in an efficient way when adhering to a predefined canonical ordering for the indeterminates. However, since the algorithms presented in the literature [1], [6] require at best $O(mn \log n)$ for the sorting effort for operands containing m and n terms respectively, Gustavson and Yun [3] proposed the generation of a nonsorted product, which could be done in $O(mn)$ operations. We shall only consider algorithms for the generation of a sorted product.

Since order of magnitude is not the sole criterion for efficiency, attempts have been made to construct more efficient algorithms for certain problems which may frequently occur in practice. We shall present results of our comparative study of these algorithms.

When starting with ordered operands, the product matrix has a certain structure, which is a tableau in reverse order, as Horowitz [5] pointed out. His algorithm SORT-MULT, which requires $O(n^2 \log n)$ operations, will be compared with our algorithm GEN-MULT, which is also based on the tableau properties. GEN-MULT's better behavior for the test cases presented by Horowitz is due to the incorporation of the tableau properties in a global way.

However, ALTRAN's original algorithm and the improved HEAP version were in fact also based on the tableau properties. Difficulty in handling equal exponents led to the design of the List-Insertion or LI algorithm. Its better performance for a large class of problems is also based on the structure of the product matrix, as will be explained by us in § 5.

Being aware of the importance of exploiting the information contained in both operands, while at the same time realizing that GEN-MULT only exploits the structure of a *general* tableau, we looked for a scheme based on the minimal information, which

* Received by the editors May 23, 1978. This work was supported by PHS Program Project Grant No. HE11310 and a Medical Center Faculty Research Grant.

† Department of Physiology and Biophysics and Department of Computer and Information Sciences, University of Alabama, Birmingham, Alabama 35294.

would be a guideline for a more efficient approach. A geometric picture of the relative location of the product exponents was obtained, from which it was seen that the product terms can be sorted in separate parallel slices. The SLICE algorithms are based on this premise. For a large class of problems of moderate size, which were presented by Johnson [6], the simple SLICE-S algorithm gave a ratio 1:4 in performance as an average relative to the LI algorithm.

For large, sparse problems an analysis of the sparse random case was made, which resulted in an optimized version, projected in algorithm SLICE-O. The average complexity for the sorting procedure is $O(mn)$ for its major operations. The factor of proportionality is dependent on the number of equal exponents in the product and on environmental conditions.

From a theoretical point of view, SLICE-O could be modified to the extent that it works exactly in linear time, which means that the execution time is proportional with nm , at the expense of auxiliary storage of the same size as the $m \times n$ product. This approach has been recently implemented as the bucket-sort algorithm by Teer [14] in the FORMULAPASCAL system. The complexity of " $X + Y$ sorting", which is the term for the mathematical abstraction of the multiplication problem was studied by Harper, Payne, Savage and Straus [4]. Their main results were that any algorithm for binary sorting (taking for convenience $n = m$) cannot perform better than $n^2 \log_2 n$, when making only use of the tableau properties. This lower bound is sharp. When taking into account the special properties of the product matrix and calculating an upper bound for the number of order types in the product, assuming that X and Y are already sorted, the lower bound for the complexity of sorting $X + Y$ is $\leq 8n \log_2 n$. On the basis of this result Fredman [2] challenges us by means of a nonconstructive proof with the assertion that any $X + Y$ problem can be sorted in $O(n^2)$.

The test data presented were obtained on an IBM 370/158 computer with MVS operating system from FORTRAN coded programs (G1 compiler), with the aid of our VARLIST list processing system [10], [11]. Only univariate polynomials were considered with single word length integer coefficients, represented in distributive form [8], [9].

2. Notation.

Polynomial operands. In the univariate case the input polynomials are represented by

$$A(x) = \sum_{1 \leq j \leq n} a_j x^{\alpha_j} \quad \text{and} \quad B(x) = \sum_{1 \leq j \leq m} b_j x^{\beta_j}.$$

α_j and β_j are positive integers which are decreasing with increasing j ; a_j and b_j are numerical coefficients.

pp-matrix. The $n \times m$ matrix of the exponent set $\{e_{ij}\}$ of the product terms $p_{ij} = a_i b_j x^{\alpha_i + \beta_j}$ ($i = 1, \dots, n$; $j = 1, \dots, m$), $e_{ij} = \alpha_i + \beta_j$,

$$\begin{vmatrix} e_{11} & e_{12} & \cdots & e_{1m} \\ e_{21} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ e_{n1} & \cdot & \cdot & \cdot & \cdot & e_{nm} \end{vmatrix}$$

FIG. 1. The *pp*-matrix.

is a special case of a tableau (see § 3.1). It is called, following Horowitz, the *pp*-matrix.

Covering set. A set of product terms p_{ij} which has the property that product terms not calculated yet have lower ordering than the largest of this set is called a "covering set" for the *pp*-matrix.

Structure number S. In case the exponent differences of the operands are random, the largest of these differences is called, following Johnson [6], the structure number for this particular random case.

PPL. This is the abbreviation for partial product list, which is an ordered subset of the product terms.

FINL. This is the abbreviation for final product list; the sorted terms which are no longer needed for the sorting of the remaining terms are deleted to FINL.

3. The algorithms GEN-MULT and SORT-MULT.

3.1. GEN-MULT. In early years of symbolic mathematics at our Institution, when computer memory was small, an algorithm was developed which was based on the properties of the matrix of the product exponents (Fig. 1). Horowitz [5] was the first to observe that this matrix is a tableau in reverse order; the integers in each row are decreasing with increasing row number, and the same property holds relative to the columns. His referral to this matrix as the *pp*-matrix will be adhered to in the sequel.

The algorithm GEN-MULT generates the terms from certain diagonal sections of the *pp*-matrix (see example in Fig. 2). One should bear in mind that this matrix is only a frame of reference as to the order in which product terms should be produced from the

1	2	3	5	6	8	9
120	<u>116</u>	<u>97</u>	90	<u>75</u>	63	56
2	4	5	7	8	9	10
<u>112</u>	108	<u>89</u>	82	67	60	48
4	6	7	9	9	11	11
<u>95</u>	91	<u>72</u>	65	<u>50</u>	43	<u>31</u>
6	6	9	9	11	11	13
90	86	67	60	45	38	26
6	7	9	10	11	12	14
77	73	54	47	32	<u>25</u>	<u>13</u>
7	8	10	11	12	14	16
74	70	51	44	29	22	<u>10</u>
8	10	10	12	13	15	17
<u>64</u>	60	<u>41</u>	34	<u>19</u>	<u>12</u>	0

FIG. 2. *pp*-matrix used as a reference for the algorithm GEN-MULT. The terms calculated in a particular step are indicated by the corresponding step number in the upper right corner. The underlined elements are the reference terms for the next step.

operands $A(x)$ and $B(x)$. At a certain stage in the process certain terms have been calculated and put in order. Realizing that the exponents form a tableau, the left upper corner represents those terms. Part of the calculated terms, no longer needed for the sorting process of the remaining terms, has been deleted to the final product list. Those which are needed for the proper continuation of the process form the so-called "covering set". They are the sorted terms of the linear partial product list PPL. In the next step PPL is updated relative to the term with lowest ordering, which is called the reference term, p_{ref} , for this step; the *pp*-matrix is scanned in increasing order of the rows, inserting terms on a particular row until the current term has lower or equal ordering than p_{ref} , or until the column number of its right neighbor equals that of a term, earlier processed in this step, which had lower or equal ordering relative to p_{ref} . As soon as updating is achieved, the top part of PPL up to the reference term is deleted to FINL.

The formal outline of the algorithm GEN-MULT, presented in Fig. 4, will be clarified by the example of the polynomials.

$$A(x) = x^{56} + x^{48} + x^{31} + x^{26} + x^{13} + x^{10} + 1,$$

$$B(x) = x^{64} + x^{60} + x^{41} + x^{34} + x^{19} + x^{12} + 1.$$

The *pp*-matrix associated with it, is pictured in Fig. 2.

Step	Start	End	Comparisons	Step	Start	End	Comparisons
1	120		0	10	60	60	4
2		<u>116</u> 112	1		56	56	
3	112	<u>112</u> 97	1		54	54	2
4	97	108 <u>97</u> 95	1 1		50	51	
5	95	<u>95</u> 90 89	1 2			<u>50</u> 48	1
6	90 89	91 90 <u>89</u> 86 77 75	2 2 2 3 1			47	2
7	86 77 75	86 82 77 <u>75</u> 74 73 72	3 2 2 1			41	3
8	74 73 72	74 73 <u>72</u> 70 68 67 64	2 1 2 4	11	48 47 41	48 47 45 44 43 <u>41</u> 38 32 31	3 3 2 2 3 1
9	70 68 67 64	70 68 67 65 <u>64</u> 60 56 54 50	3 2 2+2 1 4 3	12	38 32 31	38 34 32 <u>31</u> 29 25	3 2 1
				13	29 25	29 26 <u>25</u> 19	2 1
				14	19	22 <u>19</u> 13	1 1
				15	13	<u>13</u> 12	1
				16	12	<u>12</u> 10	1
				17		0	0
Total comparisons:							90

FIG. 3. Listing of steps for example in Fig. 2.

The total number of exponent comparisons for sorting the total product was 90, as can be verified from the steplisting in Fig. 3. The algorithm was mentioned in [8] and was documented with respect to its major aspects in [7].

Procedure GEN-MULT:

[$m \geq n > 2$] [COL(i) is column number of term on row i to be processed next] [calculate p_{11} and delete to FINL] [calculate p_{12} and put on PPL] [CMIN is the reference column number; terms with $j = \text{CMIN}$ do not have to be processed in current step] [ITOP is the first row which has not been fully processed] [main loop] [p_{ij} is calculated and inserted in PPL] [next row] [end main loop] [delete top part of PPL; generate produces i_{ref} and j_{ref}, row resp. column number of bottom term p_{ref}]	init; COL(i) $\leftarrow 1$; [$i = 2, \dots, n$] delete (p_{11}); enter (p_{12}); CMIN $\leftarrow 2$; COL(1) $\leftarrow 3$; ITOP $\leftarrow 1$; $e_{\text{ref}} \leftarrow e_{12}$; $i \leftarrow 2$; while (terms are left) do; while ($i \leq n$ and COL(i) < CMIN) do; $j \leftarrow \text{COL}(i)$; insert ($p_{ij}$); if COL($i$) = m then ITOP $\leftarrow$ ITOP + 1; COL(i) $\leftarrow$ COL(i) + 1; $\text{trm} \leftarrow e_{ij}$; if ($\text{trm} \geq e_{\text{ref}}$ or COL(i) = CMIN) then do; $i \leftarrow i + 1$; if $\text{trm} \geq e_{\text{ref}}$ then CMIN $\leftarrow j$; end; if (CMIN = 1 or $i = n$) then do; delete (TOP); generate (e_{ref}); CMIN $\leftarrow m + 1$; $i \leftarrow \text{ITOP}$; if $i_{\text{ref}} = \text{ITOP}$ then do; CMIN $\leftarrow j_{\text{ref}}$; $i \leftarrow i + 1$; end; else $i \leftarrow i + 1$; end end end end GEN-MULT
---	---

FIG. 4. Outline of GEN-MULT. Procedure **insert** inserts the term in PPL, starting at the reference term, going either upward or downward until its proper position is found.

3.2. SORT-MULT, Horowitz's best algorithm for polynomial sorting. In this algorithm [5, p. 458] a set of product terms is generated (and maintained as a binary tree structure) with the property that at most one term from each row of the pp -matrix is represented; if the column number of a possible candidate in row i is represented on the list (by a term with row $i' < i$) then this term will not be inserted in this step. Consequently this set is called a minimal covering set relative to the term with highest ordering. When updating is completed, the term with highest ordering is deleted and appended to the final product. Insertion of a candidate into the tree occurs from a point which is randomly selected.

In Table 1 (columns 2 and 4) the exponent comparisons are listed for the most relevant "random case," which is defined as follows: the exponents α_j and β_j of the polynomials $A(x)$ and $B(x)$ are generated according to the rule

$$\begin{aligned} \alpha_n &= 0; & \alpha_{n-j} &= \alpha_{n-j+1} + \text{RAND}(20) & (j = 1, \dots, n-1) \\ \beta_m &= 0; & \beta_{m-j} &= \beta_{m-j+1} + \text{RAND}(20) & (j = 1, \dots, m-1) \end{aligned}$$

where RAND(20) is an integer randomly selected from the set $\{1, 2, \dots, 20\}$. The structure number S is 20 in this case.

TABLE 1

Comparison of Horowitz's SORT-MULT with GEN-MULT and SLICE-S, for the random case with structure number 20, for polynomial operands with number of terms $n = m$. The values in the second column are taken from [5]. Although SLICE-S is more efficient, GEN-MULT will find application in case of multivariate operands with a large number of indeterminates, because SLICE-S requires isomorphic conversion of the exponent sets to their univariate images.

	SORT-MULT	GEN-MULT		SLICE-S
n	Exponent comparisons	Steps	Exponent comparisons	Exponent comparisons
3	11	4	5	0
4	33	7	15	1
5	53	10	31	3
6	107	12	50	7
7	157	14	79	16
8	221	16	114	28
9	264	18	159	42
10	387	21	208	58
11	554	23	273	82
12	668	25	343	110
13	757	26	413	141
14	1053	27	508	179
15	1118	29	600	213
16	1409	31	720	251
17	1596	32	855	299
18	1643	34	951	364
19	2289	36	1103	394
20	2031	38	1268	489

3.3. Comparison of GEN-MULT and SORT-MULT for Horowitz's input polynomials. The test data which Horowitz presented [5, p. 461] were classified as "dense," "sparse", "intermediate" and "random". However, the distinction "structured" and "unstructured" would have been more meaningful as a basis for comparison of performance. "Structured" could be defined as a regular pattern for the exponent sequences of either operand. Any algorithm that takes advantage of the structure of the problem is bound to perform better than algorithms which are not designed on this basis. This fact becomes clear when inspecting Horowitz's test data for his SORT-MULT algorithm against conventional methods, which have the same upper bound $O(mn \log n)$ for the sorting effort.

GEN-MULT is an improvement over SORT-MULT due to the global incorporation of the structure. For the "random case" with $S = 20$ an average ratio 0.52 was found in favor of GEN-MULT as can be verified by inspecting the data in columns 2 and 4 of Table 1. Although the linear insertion gives an a priori $O(mn^2)$ bound for the sorting effort of GEN-MULT, as opposed to SORT-MULT's bound $O(nm \log n)$ due to the binary search strategy, a consistent increase in ratio with increasing n was not observed for the low values of n in the test data. With the availability of the SLICE algorithms, the application of GEN-MULT will be restricted to special multivariate problems in which n and m are of moderate size. In § 6 a review is given.

The characteristics and differences of the algorithms are listed below.

GEN-MULT

SORT-MULT

I. CHARACTERISTICS

- | | |
|---|--|
| <ol style="list-style-type: none"> 1. A partial product list, maintaining a linear list structure, is updated by inserting terms from the <i>pp</i>-matrix, relative to the term which is <i>at the bottom</i> of the list after completion of the previous step. 2. A step terminates by deleting the top part of the partial product list up to the reference term. | <ol style="list-style-type: none"> 1. A partial product list, in the form of a binary tree, is updated <i>relative to the top term</i> until each row of the <i>pp</i>-matrix either has a representative, or if it is known that the current candidate has lower ordering than the top term. 2. A step terminates by deleting the top term from the list. |
|---|--|

II. DIFFERENCES

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. The algorithm incorporates dynamically <i>the global structure</i> of the <i>pp</i>-matrix, since it may occur that several closely spaced terms on a particular row are inserted in one step. 2. The number of steps is $O(n + m)$. 3. Although theoretically the same upperbound $O(mn \log_2 n)$ holds for the number of exponent comparisons when the partial product list would be maintained as a tree structure, the cheaper linear list structure proved sufficiently adequate to bring out GEN-MULT's superior performance for the test cases presented by Horowitz. 4. The partial product list contains $O(2n)$ terms. | <ol style="list-style-type: none"> 1. The algorithm looks for the largest exponent of a minimal set, which means that only <i>the local structure</i> of the <i>pp</i>-matrix is taken into account. 2. The number of steps ranges from $O(n + m)$ for the dense case to $O(nm)$ for the sparse case. 3. The number of exponent comparisons was proven to be $O(mn \log_2 n)$. 4. The partial product list contains a number of terms which is $\leq n$. |
|--|--|

4. The algorithms SLICE-S and SLICE-O.

4.1. The 2-dimensional picture of the relative location of the product exponents.

We have seen in § 2 that GEN-MULT exploits the properties of a general tableau. However, the *pp*-matrix has additional properties, namely the difference of two elements in the same row only depends on their column numbers; a similar statement holds for 2 elements in the same column. In formula:

$$e_{ij} - e_{ik} = \alpha_i + \beta_j - \alpha_i - \beta_k = \beta_j - \beta_k,$$

$$e_{ij} - e_{kj} = \alpha_i + \beta_j - \alpha_k - \beta_j = \alpha_i - \alpha_k.$$

This means that we do not need all the information contained in the *pp*-matrix; the order of the product terms is completely defined by the two linear arrays v_{1j} and v_{i1} of the exponent differences of each of the operands. If one defines:

$$v_{11} = 0; \quad v_{1j} = e_{11} - e_{1j} \quad (j = 2, \dots, m); \quad v_{i1} = e_{11} - e_{i1} \quad (i = 2, \dots, n)$$

it follows $v_{1i} - v_{1j} = \beta_j - \beta_i$; $v_{i1} - v_{j1} = \alpha_j - \alpha_i$.

The exponent differences v_{i1} and v_{1j} are plotted on the axes ξ and η of a Cartesian coordinate system, such that $\xi_{i-1} \equiv v_{i1}$ ($i = 2, \dots, n$), $\eta_{j-1} \equiv v_{1j}$ ($j = 2, \dots, m$). When assigning to the points v_{ij} (with coordinates ξ_{i-1}, η_{j-1}) the norm $|v_{ij}| = \xi_{i-1} + \eta_{j-1}$, then these points can be considered the images of the product exponents in the sense that the following relationships are valid:

1. $e_{ij} - e_{kl} = -|v_{ij}| + |v_{kl}|$;
2. $e_{ij} = e_{kl}$ iff $\xi_{i-1} + \eta_{j-1} = \xi_{k-1} + \eta_{l-1}$, i.e. if the corresponding points v_{ij}, v_{kl} are located on the straight line $\xi + \eta = c$, such that $c = \xi_{i-1} + \eta_{j-1}$.
3. $e_{ij} > e_{kl}$ iff $\xi_{i-1} + \eta_{j-1} < \xi_{k-1} + \eta_{l-1}$.

These formal results can be stated in the following simple manner:

When moving a rod from the origin to the opposite end of the rectangle under equal angles with the coordinate axes one encounters the images of the product exponents in their correct ordering.

4.2. Algorithm SLICE-S. In a first intuitive approach, presented by algorithm SLICE-S, the rectangle is divided in (at most) $2(n + m) - 6$ slices by choosing the set of parallel lines

$$\xi + \eta = v_{j1} \quad (j = 2, \dots, n); \quad \xi + \eta = v_{1j} \quad (j = 2, \dots, m);$$

$$\xi + \eta = v_{n1} + v_{1j} \quad (j = 2, \dots, m-1);$$

$$\xi + \eta = v_{1m} + v_{j1} \quad (j = 2, \dots, n-1)$$

as shown in Fig. 5. The marked reduction in number of symbolic comparisons is obtained at the expense of a number of numerical comparisons for each element whether or not it is located in the current slice. This implies that the linear arrays v_{1j} and v_{j1} actually have to be constructed as the initializing step.

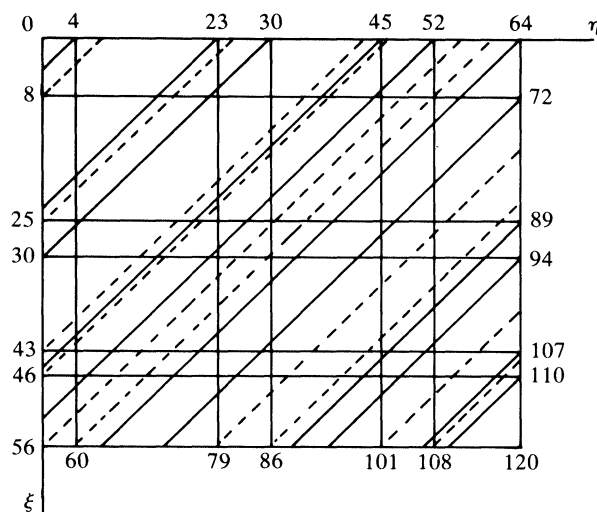


FIG. 5. The pp -matrix of Fig. 2, pictured relative to the value of the exponents. The points on the coordinate axes represent the partial sums of the exponent differences of the polynomials $B(x)$ and $A(x)$ from the example. Only the internal points of the rectangle which are located in one "slice" (section between adjacent parallel lines) have to be put in order on PPL. Following procedure PUTLIST one can verify that 14 symbolic exponent comparisons are required, when comparison starts from the bottom of PPL. It was earlier found that the same example requires 90 comparisons for GEN-MULT.

In Table 1 the symbolic comparisons for SLICE-S for Horowitz's input polynomials for the random case are listed in the 4th column.

4.3. Optimization for the sparse, random case implemented in algorithm SLICE-O.

4.3.1. Division of the rectangle. In an attempt to find the optimal approach, we present the following lemmas.

LEMMA 1. *The division of the rectangle in slices of equal area will ensure minimization of the symbolic sorting effort, when averages are taken over a large number of tests.*

Proof. The points of the rectangular network are distributed with uniform density, when averages are taken over a large number of tests, in which n , m and S are kept constant. Equal partitioning of the rectangle as a means for optimization is a direct consequence of a simple calculation from the calculus of variations; since the sorting for the j th section, containing x_j terms, requires $O(x_j^2)$ (in case of a linear list structure), the sum $S = \sum_{1 \leq j \leq T} x_j^2$ has to be minimized under the constraint $\sum_{1 \leq j \leq T} x_j = \text{constant}$ ($=nm$). It is readily found that $x_1 = x_2 = \dots = x_T$, in which T is the total number of sections.

4.3.2. The rigorous approach. Once the constant of proportionality c in the relationship $T = cmn$ for a particular computing environment is obtained (see § 4.3.6), the optimal value for T can be calculated. The division of the rectangle in slices of equal area then involves the following calculations. Referring to the notation of the outline in Fig. 6, we define

$$a = VA(n) \quad (\text{degree of } A(x)); \quad b = VB(m) \quad (\text{degree of } B(x))$$

and assume that $b \geq a$. Dividing the rectangle into two congruent triangles and the parallelogram which they enclose, we introduce

Procedure SLICE-S:

[choose $m \geq n > 2$]

[partial sums of exp. diff. of $A(x)$]

[partial sums of exp. diff. of $B(x)$]

[column number of next term on row i]

[main loop in PUT-LIST starts with row I]

[p_{11} has highest ordering]

[since $e_{22} < e_{12}$ and $e_{22} < e_{21}$, initialization comprises deletion of terms p_{1j} and p_{i1} up to $\min(p_{12}, p_{21})$]

[TERM1 and TERM2 are terms at the boundary; side1 = true if V1 moves along left vertical boundary; side2 = true if V2 moves along upper horizontal boundary]
[DIAG = min(V1, V2). At start V1 moves along left vertical boundary, V2 along upper horizontal boundary]

VA(1) $\leftarrow$ 0; VA(i) $\leftarrow$ v_{i1} ; [$i = 2, \dots, n$]

VB(1) $\leftarrow$ 0; VB(j) $\leftarrow$ v_{1j} ; [$j = 2, \dots, m$]

COL(i) $\leftarrow$ 2; [$i = 2, \dots, n-1$]

$I \leftarrow 2$;

delete (p_{11}); $i \leftarrow 2$; $j \leftarrow 2$;

init: begin;

while VA(2) > VB(j) **do**; **delete**(p_{1j}); $j \leftarrow j+1$; **end**;

while VA(i) < VB(2) **do**; **delete**(p_{i1}); $i \leftarrow i+1$; **end**;

if VA(i) < VB(j) **then do**; **delete**(p_{i1}); $i \leftarrow i+1$; **end**;

else do;

if VA(i) > VB(j) **then do**; **delete**(p_{1j}); $j \leftarrow j+1$; **end**;

if VA(i) = VB(j) **then do**; TERM = sum(p_{i1}, p_{1j});

delete(TERM); $i \leftarrow i+1$; $j \leftarrow j+1$;

end;

end;

end;

V1 $\leftarrow$ VA(i); TERM1 $\leftarrow$ p_{i1} ; V2 $\leftarrow$ VB(j); TERM2 $\leftarrow$ p_{1j} ;

side1 $\leftarrow$ true; side2 $\leftarrow$ true; DIAG $\leftarrow$ 0;

main loop:

while

DIAG $\leq$ min(VA(n) + VB($m-1$), VB(m) + VA($n-1$)) **do**

if V1 < V2 **then do**; DIAG $\leftarrow$ V1; TERM $\leftarrow$ TERM1; **end**;

else do; DIAG $\leftarrow$ V2; TERM $\leftarrow$ TERM2; **end**;

if V1 = V2 **then** TERM $\leftarrow$ sum(TERM1, TERM2);

put-list(DIAG); **append**(TERM);

delete (PPL);

[terms above the line $\xi \times \eta = \text{DIAG}$ are inserted in PPL; after the terms on DIAG have been contracted and appended to PPL, PPL is appended to FINL]
 [next value for V1]
 [initialize moving along lower horizontal boundary]
 [next value for V2]
 [initialize moving along right vertical boundary]

```

if(V1 ≤ V2 and side1 = true) then do;
if i < n then do; i ← i + 1; V1 ← VA(i); TERM1 ← pi1;
    end; else do; side 1 ← false; i ← 2;
    end; end;
if(V1 ≥ V2 and side2 = true) then do;
if j < m then do; j ← j + 1; V2 ← VB(j); TERM2 ← p1j;
    end; else do; side2 ← false; j ← 2;
    end; end;
if(V1 ≤ V2 and side1 = false) then do;
V1 ← VA(n) + VB(i); TERM1 ← pni; i ← i + 1;
    end;
if(V1 ≥ V2 and side2 = false) then do;
V2 ← VB(m) + VA(j); TERM2 ← pjm; j ← j + 1;
    end;
delete(remaining terms);
end SLICE-S;
end;

```

Procedure PUT-LIST:

```

init;
THRU ← false; THR(i) ← false; [i = 1, ..., n - 1]
while THRU = false do; THRU ← true; i ← I; CMIN ← m;
main loop:
while(i ≤ n - 1 or CMIN > 2) do;
if THR(i) = false then do; j ← COL(i);
if vij < DIAG then do; insert(pij);
    if COL(i) = m - 1 then I ← I + 1;
    else do; COL(i) ← COL(i) + 1;
        vij ← VA(i) + VB(j); THRU ← false;
    end; end;
if vij = DIAG then TERM ← sum(TERM, pij);
if vij ≥ DIAG then do; CMIN ← j; THR(i) ← true;
    end; end; i ← i + 1;
end;
end PUT-LIST;

```

[THRU is true if all terms above DIAG have been processed;
 THR(i) is true if next term on row i is below DIAG]
 [in each cycle only one term per row is inserted, because diagonally located terms tend to be closer spaced than terms on the same row]

FIG. 6. Outline of procedure SLICE-S with subprocedure PUT-LIST.

S_{tr} = the number of slices of each triangle,

S_p = the number of slices of the parallelogram.

S_{tr} and S_p can be solved from $T = 2S_{tr} + S_p$ (total number of slices) and $0.5a^2/S_{tr} = (b-a)a/S_p$ (equal area for trapezoidal and parallelogrammic slices). It follows $S_{tr} = 0.5Ta/b$; $S_p = T(b-a)/b$. The increments of DIAG (see outline in Fig. 6) in the triangular sections are dependent on the slice number and are denoted by s_j ($j = 1, \dots, S_{tr}$). With the notation $U \equiv a/S_{tr}^{0.5}$ it is found after a simple calculation

$$(1) \quad s_j = U(j^{0.5} - (j-1)^{0.5}) = U/(j^{0.5} + (j-1)^{0.5}) \quad (j = 1, \dots, S_{tr}).$$

Provided that $S_p \geq 1$, the equal increments s_p of the parallelogrammic area are b/T .

4.3.3. Relaxing the condition (1). For each individual random case the network vertices are nonuniformly distributed over the rectangular area. The implementation of (1) would not be practical, due to the great amount of square root calculations. By means of the following considerations we virtually eliminate the cumbersome computations, but are globally upholding the requirement of equal area for the separate slices. It is first noticed in Lemma 2.

Procedure SLICE-O;

[S_{tr} is number of slices for triangle]

[I_m is number of main intervals of equal length
 M_{inc} , the main increment, which is the length
of the interval between the j^2 th and $(j+1)^2$ -
th diagonal]

[each main division point $i (=j^2)$ is the center
of $2i$ subintervals of length $M_{inc}/(2j)$; these
are the subincrements S_{inc}]

[last increment L_{inc} equals M_{inc} divided by
 $2 \cdot I_m$. These increments are added to DIAG,
starting at the center of the I_m th interval until
a total length a]

[S_p is number of slices in parallelogram]

[s_p are equal increments]

[algorithm for second triangle equals that of
first triangle in reverse order]

```

a ← min(VA(n), VB(m));
b ← max(VA(n), VB(m));
T ← total number of slices;
Str ← 0.5 * T * a/b;
Im ← Str0.5; Minc ← a/Im;
first triangle:
DIAG ← Minc; put-list(DIAG);
DIAG ← DIAG + 0.5 * Minc; put-list(DIAG);
STEPS ← Im - 1;
main loop:
for j ← 2 to STEPS do;
j' ← 2j; Sinc ← Minc/j';
for k ← 1 to j' do; DIAG ← DIAG + Sinc;
put-list(DIAG); end;
end;
remaining slices:
Linc ← Minc/(2 * Im);
STEPS ← Im + (a - Im * Minc)/Linc;
for k ← 1 to STEPS do;
DIAG ← DIAG + Linc; put-list(DIAG);
end;
parallelogram:
if a = b then goto last-triangle;
Sp ← T(b - a)/b; sp ← b/T;
if (Sp ≤ 1 or sp = 0) then go to next-vertex;
for k = 1 to Sp do;
DIAG ← DIAG + sp; put-list(DIAG);
end;
next vertex: DIAG ← b; put-list(DIAG);
last-triangle:
end SLICE-O;
```

FIG. 7. Outline of the main aspects of algorithm SLICE-O. For greater clarity we have omitted the obvious actions at each step, i.e. initializing TERM (= terms located on DIAG; see PUT-LIST, Fig. 6) as an empty list and, after insertion is completed, appending TERM to PPL. Then PPL is deleted to FINL.

LEMMA 2. The sum of the increments between the j^2 -th and the $(j+1)^2$ -th slice is independent of j and thus equals $U \equiv a/S_{tr}^{0.5}$.

Proof. It follows from (1) that

$$\sum_{j^2+1 \leq i \leq (j+1)^2} U(i^{0.5} - (i-1)^{0.5}) = U(-j + (j+1)) = U.$$

Consequently, after dividing the triangle side in $S_{tr}^{0.5}$ equal main intervals U , we now take $U/(2j)$ for the slice width around the j th main interval. Since $j^2 = i$, where i numbers the subintervals, the exact slice width $U/(i^{0.5} + (i-1)^{0.5})$ at the j th main division point is approximated by $U/(2i^{0.5})$. Thus, instead of dividing the trapezoidal sections between the j^2 and $(j+1)^2$ diagonals into $2j+1$ sections of equal area, this section is divided into j sections of equal slice length $U/(2j)$ and $j+1$ sections of equal slice length $U/(2j+2)$.

4.3.4. Outline of algorithm SLICE-O. The simplified version of the rigorous approach, discussed above, is outlined in Fig. 7.

4.3.5. Order of magnitude for the sorting effort.

The essential operations.

LEMMA 3. *Asymptotic optimal performance of SLICE-O for the sparse random case, for its major operations, occurs when the total number of slices T is chosen proportional with mn ; $T = cmn$, where c depends on the degree of sparseness and on environmental conditions. This implies that the algorithm performs in linear time and requires $O(mn)$ for the symbolic sorting effort.*

Proof. The major operations of the algorithm are

1. The numeric operations for calculating DIAG, which are $O(T)$.
2. The symbolic sorting effort per term for the terms of each slice, which, in our case of linear insertion, requires $O(nm/T)$. Accordingly, the total effort is $O(n^2m^2/T)$.
3. Concatenation of the sorted PPL's which requires $O(T)$ operations.

It follows that if and only if T is chosen proportional with nm , the asymptotic behavior of the algorithm is $O(mn)$.

Consequently, the total symbolic sorting effort is $O(mn)$ and the total computing time is proportional with mn .

The minor operations. An operation not included in the proof above is the assignment of the correct slice to each product term. It is considered a minor operation in the area of symbol manipulation due to its low weight factor. In the outline of algorithm PUT-LIST (Fig. 6) the value v_{ij} (see § 4.1), corresponding with a candidate product term p_{ij} , has to be compared with the slice boundary value DIAG. Since $T = O(mn)$, while only m terms in each row are placed in a slice, the number of vain comparisons per term is $O(n)$. This accounts for the nonlinearity of the total procedure. Other minor operations in PUT-LIST have the same bound.

Theoretical improvement of the term insertion. The term insertion could be reduced to linear behavior when generating the T PPL's simultaneously and calculating for each term its appropriate slice. The linearity achieved in this manner would be at the cost of auxiliary memory space, which in our system, for $n = m = 300$, would be approximately 180K bytes.

An intermediate approach, which requires approximately $\frac{1}{10}$ of the auxiliary space mentioned above for the same example, would be to generate the $T^{0.5}$ major PPL's (with equal slice width (see § 4.3.3)) sequentially, but generating the $2j$ minor PPL's within each major slice simultaneously. With 1 term per row as an average located in a major slice, the insertion of this term in the correct minor slice would be bounded by $\log_2 n$, when applying binary search.

4.3.6. Experimental results.

Tests with algorithm SLICE-O. Tests with SLICE-O were done for $n = m$, with values 50, 70, 90, 150, 200, 300 and S -values 64 and 1000. The definition of the structure number S was given in § 2 and § 3.2. 64 was the largest value for S employed in Johnson's experiments. From observation it was found that for $S = 64$ approximately $\frac{2}{3}$ of the nm product terms are distinct. $S = 1000$ corresponded with 98% sparsity. Therefore this value for S was used to generate operands which were good representations of the completely sparse random case. In Table 2 are displayed the symbolic comparison effort per term, the value of c (see § 4.3.5) which gives the ratio of the optimal number of slices T versus n^2 . The observed minimal time divided by n^2 is listed in the last column. The data listed for optimal T were derived from a great number of tests over a certain range of T values, such that a significant increase in time was observed at both ends of the spectrum.

With respect to the data, the following comments are made:

1. The uncertainty in symbolic sorting effort (which is closely related to that in

TABLE 2

Data obtained with algorithm SLICE-O for S -values 64; 1,000; 10,000. In the first column the value of $n (=m)$ is displayed. The number of symbolic sorting operations per term is shown in the 2nd column. From the factor of proportionality c the optimal value of T can be derived. The data in the 4th column show that the algorithm exhibits near linear behavior.

S = 64				S = 1,000			
n	Symb. sort/ n^2 $\pm 10\%$	$c = T/n^2$ $\pm 10\%$	Time/ n^2 (millisec) $\pm 2\%$	n	Symb. sort/ n^2 $\pm 10\%$	$c = T/n^2$ $\pm 10\%$	Time/ n^2 (millisec) $\pm 2\%$
50	2.2	0.12	2.5	50	2.3	0.16	3.1
70	2.7	0.08	2.5	70	2.6	0.12	3.1
90	2.8	0.06	2.4	90	3.0	0.10	3.1
150	2.8	0.05	2.5	150	3.2	0.08	3.3
200	2.9	0.05	2.4	200	3.3	0.08	3.4
300	3.3	0.05	2.6	300	3.5	0.07	3.5

S = 10,000			
n	Symb. sort/ n^2 $\pm 10\%$	$c = T/n^2$ $\pm 10\%$	Time/ n^2 (millisec) $\pm 2\%$
150	3.1	0.10	3.3

T/n^2) is much larger than the uncertainty in computing time. This is due to the low weight factors of the operations which are linearly dependent on T as specified in § 4.3.5. In mathematical terms this can be seen as follows. When defining $y := \text{total effort}/n^2$, $x := T/n^2$, we find $y = 1/x + (c_2 + c_3n)x/c_1 + \dots$ (terms of lower order in x) where c_1 and c_2 are the weight factors for the operation 2 and 1 + 3 respectively c_3 is the weight factor for term comparison operations.

The significant part of this equation represents the branch of a nonorthogonal hyperbola in the first quadrant with asymptotes $x = 0$ and $c_1y = (c_2 + c_3n)x$. Due to the small ratio $(c_2 + c_3n)/c_1$, the slope of this line is small. From this it follows that the x -value, corresponding with minimal y , is not sharply defined.

The uncertainty in time is mainly due to fluctuations in observed CPU time for identical runs, which is inherent with the MVS operating system (for the IBM 370/158).

2. The effect of the nonlinear term on time is not noticeable in the case $S = 64$. In the completely sparse case there is an increase of approximately 10% in time. Since it is most likely that the range in problem size of our test runs covers all cases occurring in the field of symbolic computation, we did not feel the need to implement one of the alternative approaches mentioned in § 4.3.5.

3. The ratio 2.5/3.3 in time of the cases characterized by $S = 64$ and $S = 1000$ is in close agreement with the observed ratio 2/3 for the number of distinct terms versus the size of the total product; the slightly larger ratio 2.5/3.3 is due to the fact that equal exponents of terms on an unsorted PPL will in general not occur sequentially.

4. In order to verify that the sorting effort is only dependent on the degree of sparsity, we listed at the bottom of Table 2 the results of a few tests with $S = 10^4$, for $n = 150$.

5. Due to the low average number of terms per slice, which is $1/c$ (see Table 2), we do not expect great advantage in time when employing a different list structure.

Comparative data for SLICE-S and SLICE-O. More extensive data for SLICE-S will be presented in § 5.4, after discussion of the LI algorithm. The disadvantage of SLICE-O is that an a priori guess of the optimal number of slices has to be made, which means that the system constant c has to be determined. The simple SLICE-S algorithm will cover a large class of problems without substantial sacrifice in computing time, as can be seen from the data in Table 3.

TABLE 3
Comparison of algorithms SLICE-S and SLICE-O. SLICE-O becomes significantly more profitable for the completely sparse case and large values for the number of terms of the polynomial operands.

	SLICE-S		SLICE-O	
	S = 64			
<i>n</i>	Symb. sort/ <i>n</i> ²	Time/ <i>n</i> ² (millisec)	Symb. sort/ <i>n</i> ²	Time/ <i>n</i> ² (millisec)
50	3.5	2.7	2.2	2.5
70	4.1	2.7	2.7	2.5
90	4.6	2.8	2.8	2.4
S = 1,000				
50	4.8	3.9	2.3	3.1
70	6.4	4.3	2.6	3.1
90	7.8	4.6	3.0	3.1

4.3.7. The multivariate case. Construction of the linear arrays of the exponent differences in the SLICE procedures requires that in the multivariate case ($\nu \geq 2$) the exponent sets

$$\{\varepsilon_{1j}^{(1)}, \varepsilon_{2j}^{(1)}, \dots, \varepsilon_{\nu j}^{(1)} | j = 1, \dots, n\} \text{ of polynomial } A(x),$$

$$\{\varepsilon_{1j}^{(2)}, \varepsilon_{2j}^{(2)}, \dots, \varepsilon_{\nu j}^{(2)} | j = 1, \dots, m\} \text{ of polynomial } B(x)$$

have to be mapped into the ring of the integers such that the order of the product terms produced from the univariate images uniquely defines the order of the multivariate product.

It is emphasized however that the SLICE algorithms only require knowledge of the exponent sets

$$\{\alpha_j^{(1)} | j = 1, \dots, n\} \quad \text{and} \quad \{\alpha_j^{(2)} | j = 1, \dots, m\}$$

of these images, since sorting in each slice is performed on the given multivariate operands. (The alternative approach, which requires inverse mapping of the univariate product, has been presented in a different context [13].)

Order of the sets $\{t^{(i)}\}$ of the terms of either operand is defined in the usual way by assigning a certain canonical ordering to the variables, say, $x_1 > x_2 > \dots > x_\nu$. If

$$t_1^{(i)} := c_1^{(i)} x_1^{\varepsilon_{11}} x_2^{\varepsilon_{21}} \dots x_\nu^{\varepsilon_{\nu 1}}, \quad t_2^{(i)} := c_2^{(i)} x_1^{\varepsilon_{12}} x_2^{\varepsilon_{22}} \dots x_\nu^{\varepsilon_{\nu 2}},$$

then by definition $t_1^{(i)} > t_2^{(i)}$ if and only if one of the following condition holds

- (1) $\varepsilon_{11} > \varepsilon_{12}$,
- (2) $\varepsilon_{i1} = \varepsilon_{i2} \quad (i = 1, \dots, j; j < \nu) \quad \text{and} \quad \varepsilon_{i+1,1} > \varepsilon_{i+1,2}.$

(If a variable does not explicitly appear on a term, then its exponent is interpreted as 0.)

The isomorphic mapping is achieved as follows.

First the operators $\{\rho_j | j = 2, \dots, \nu\}$ for the mapping operation are defined:

$$\rho_j := \sum_{1 \leq i \leq 2} \left(\max_{1 \leq k \leq n'} (\varepsilon_{jk}^{(i)}) - \min_{1 \leq k \leq n'} (\varepsilon_{jk}^{(i)}) \right) + 1 \quad \left(n' = \begin{cases} n & \text{if } i = 1 \\ m & \text{if } i = 2 \end{cases} \right)$$

where the maxima (and minima in case of the occurrence of negative exponents (see e.g. [8, p. 3])) for the j th variable are computed for the terms of $A(x)$ and $B(x)$ respectively.

Then the recursive generation of the integers $\beta_{kj}^{(i)}$ according to the scheme

$$\beta_{1j}^{(i)} = \varepsilon_{1j}^{(i)}; \quad \beta_{kj}^{(i)} = \varepsilon_{kj}^{(i)} + \rho_k \beta_{k-1,j}^{(i)} \quad (k = 2, \dots, \nu)$$

for each term $t_j^{(i)}$ will yield the sets $\{a_j^{(i)}\}$, with $\alpha_j^{(i)} := \beta_{\nu j}^{(i)}$, and will guarantee the required isomorphism.

5. ALTRAN's List-Insertion or LI algorithm. The advantage of the LI algorithm for problems of moderate size over alternative methods in the ALTRAN system is due to the exploitation of the structure of the partial product list, although this fact has not been clearly recognized.

5.1. Basic idea. A certain linear ordered partial product list is maintained such that all terms not processed yet have lower ordering than the current top term. This list of terms can be considered a "covering set". This property is preserved, if, after deletion of the top term, say p_{ij} , the "neighbors" $p_{i,j+1}$ and $p_{i+1,j}$ are processed in case the following conditions are fulfilled:

1. these terms have not been processed earlier;
2. the terms $p_{i-1,j+1}$ and $p_{i+1,j-1}$ have earlier been processed.

Although the LI algorithm with its linear list structure has an a priori $O(n^2 m)$ behavior as compared with $O(nm \log_2 n)$ for the HEAP algorithm [1], [6], [12] with respect to the number of exponent comparisons, LI performed considerably better than HEAP for virtually all cases which are likely to occur in applications.

5.2. Explanation of LI's superior performance. We were able to show that the effect of two lemmas are responsible for LI's better performance.

LEMMA 4. *If $e_{ij} > e_{kl}$ then the probability that $e_{i,j+1} > e_{k,l+1}$ is greater than $\frac{1}{2}$.*

Proof. $e_{i,j+1} - e_{k,l+1} = \alpha_i - \alpha_k + \beta_{j+1} - \beta_{l+1} = (\alpha_i + \beta_j) - (\alpha_k + \beta_l) + (\beta_{j+1} - \beta_j) - (\beta_{l+1} - \beta_l) = (e_{ij} - e_{kl}) + (\beta_{j+1} - \beta_j) - (\beta_{l+1} - \beta_l)$. Since $e_{ij} - e_{kl} > 0$, whereas, in the random case, the remaining expression has equal probability to be > 0 or < 0 , we conclude that $e_{i,j+1} - e_{k,l+1} > 0$ in the majority of cases.

One can express Lemma 4 in the following way: if one generates the set of right "neighbors" of the terms of a covering set, then their ordering will resemble the ordering of the former set.

LEMMA 5. *If p_{ij} and p_{kl} are terms of a covering set, such that $p_{ij} > p_{kl}$ then, if $p_{i,j+1}$ is inserted prior to $p_{k,l+1}$, the number of unsuccessful exponent comparisons is minimal if insertion starts from the bottom of PPL.*

Proof. Since, according to Lemma 4, the probability that $e_{i,j+1} > e_{k,l+1}$ is greater than $\frac{1}{2}$, no unsuccessful comparison of the corresponding terms $p_{i,j+1}$, $p_{k,l+1}$ is done in the majority of cases, when processing $p_{i,j+1}$ prior to $p_{k,l+1}$, while starting the exponent comparisons from the bottom of PPL. It is further noticed that the terms of a "covering set" are in general closer spaced than a term and its neighbor on the same row (column); the average distance of two neighbors is $N/(n+m-2)$, where N is the degree of the product. The average distance between two terms of the final product is $N/(nm)$. Since a

covering set (which we supposed to include p_{ij} and p_{kl}) can be considered a not fully updated part of the final product list, it is expected that $e_{kl} > e_{i,j+1}$ in the majority of cases, so that insertion from the bottom of PPL also minimizes the amount of unsuccessful comparisons relative to this pair.

5.3. The Quasi-LI or Q-LI algorithm. In order to compare the LI algorithm with GEN-MULT and SLICE, also relative to time, we implemented a version of LI which is conceptually simpler, the so-called Quasi-LI algorithm. Starting with a covering set, only the right neighbor(s) of the top term is (are) processed. In this way one avoids the complicated checks mentioned under 1 and 2 in § 5.1. However, due to the asymmetric approach the number of exponent comparisons is $\leq 10\%$ larger, as can be seen from Table 4.

TABLE 4

SLICE-S and GEN-MULT as compared with ALTRAN's LI algorithm and with a simpler version, Q-LI, which was implemented in order to obtain time comparisons. In the first column the number of terms $n (= m)$ is displayed, in the second the structure number S . The data in the third column are taken from [6]. The advantage in time of GEN-MULT over Q-LI is mainly due to the simpler administrative procedure of the former algorithm. SLICE-S works in linear time (for fixed S) for these input polynomials and therefore its relative efficiency increases with increasing n .

n	S	LI	Q-LI		GEN-MULT		SLICE-S	
		Exponent comp./ n^2	Exponent comp./ n^2	Time/ n^2 (millisec)	Exponent comp./ n^2	Time/ n^2 (millisec)	Exponent comp./ n^2	Time/ n^2 (millisec)
10	4	1.39	1.47	4.5	1.48	3.0	0.18	1
10	16	1.97	2.17	4.5	2.00	3.5	0.53	2.5
10	64	2.20	2.49	4.5	2.36	4.0	0.78	3.0
30	4	1.93	2.01	7.7	1.83	2.2	0.36	1
30	16	3.80	4.25	8.1	3.44	3.2	1.34	2.0
30	64	5.18	5.98	5.2	5.04	4.3	2.40	2.8
50	4	2.19	2.21	4.7	1.90	2.1	0.37	0.8
50	16	4.83	5.26	6.4	4.01	3.2	1.61	1.7
50	64	7.46	8.91	7.4	7.20	5.2	3.48	2.7
70	4	2.29	2.29	5.0	1.94	2.1	0.37	0.7
70	16	5.47	6.01	7.6	4.39	3.4	1.75	1.6
70	64	9.64	11.28	9.3	8.78	5.9	4.14	2.7
90	4	2.38	2.33	5.2	1.96	2.1	0.40	0.7
90	16	5.94	6.47	8.1	4.65	3.7	1.86	1.6
90	64	11.93	13.28	10.8	10.06	6.9	4.63	2.8

5.4. Comparison of Q-LI with GEN-MULT and SLICE-S. In Table 4 the relative behavior of the various algorithms is displayed for the test data presented by Johnson. Comparing first the symbolic sorting effort of LI (and Q-LI) with GEN-MULT it is seen that GEN-MULT is slightly more profitable. LI recruits new product terms on the basis of the structure of the previous sorted PPL.

The advantage in time of GEN-MULT is considerably greater. This can be understood when realizing that with LI, for each product term processed, one has to keep track of the corresponding row and column number(s). This even leads to the paradoxical situation (in our system) that it takes longer to process 30×30 terms with low value for S (so that many could be contracted) than the sparse case.

SLICE-S greatly profits from structured cases (low value for S). It performs better than GEN-MULT and LI with increasing value of n (keeping S fixed), since the latter

algorithms do not perform in linear time. Since SLICE-S becomes significantly worse than SLICE-O only for very large problems, SLICE-S is expected to perform in linear time (keeping S fixed) for the problems in Table 4. This expectation is confirmed by the test data.

6. Review of the presented algorithms. We have shown that GEN-MULT performed better than Horowitz's SORT-MULT and ALTRAN's LI-algorithm for the test cases presented in the respective papers. In its present form GEN-MULT's theoretical bound is $O(mn^2)$, but transfer to a different data structure could lower this bound to SORT-MULT's $O(mn \log n)$ behavior. GEN-MULT will find useful application only for multivariate polynomials with many indeterminates for which the mapping of the exponent sets required for the SLICE approach, as explained in § 4.3.7, could be cumbersome. Those polynomials, when containing a great number of terms, would be most efficiently handled in recursive representation, as reported by us in [8]. In this case multiplication is likewise done recursively, with each intermediate univariate case, which then would be of moderate size, handled by GEN-MULT.

The SLICE algorithms will be appropriate for the remaining cases, which are the univariate polynomials and the multivariate polynomials with a small (say less than 5) number of indeterminates.

SLICE-S is efficient for moderate size (say up to 100 terms) of the operands and requires $O(n)$ auxiliary space.

SLICE-O requires advance knowledge of the system constant c , although a rough estimate is sufficient for an efficient performance. In its present form it works in near linear time ($\text{time}/n^2 = \text{constant}$) as an average for the random case and requires very little auxiliary storage. Although $1/c \approx 20$ terms is the average value for the work space, in exceptional cases this number could be appreciably higher, due to situations where clusters of product terms would appear.

7. Connection with theoretical results. The $O(mn)$ performance of the SLICE algorithms and of Teer's bucket-sort algorithm [14] as an average for the random case can be considered a step forward in the design of efficient algorithms for $X + Y$ sorting. Research should be continued in order to realize the possibility of $O(n^2)$ sorting for any $X + Y$ problem. Although there is an infinite variety in exponent sequence for each of the operands, for fixed n , one should take into account that the number of order types for the product is finite (Harper, Payne, Savage and Straus [4]). The algorithm we are looking for, should be able to dynamically recognize the order type with which the problem corresponds. Furthermore, Fredman's proof [2] is based on queries for each individual product term relative to a suitably chosen integer. This implies that the approach of older algorithms, which merely utilize binary comparisons, may have to be abandoned. The SLICE algorithms could be a first step towards realizing the theoretically found $O(n^2)$ performance.

Acknowledgment. The author wants to thank the reviewers and also Frank Teer for their constructive remarks and for their referral to the literature concerning the mathematical issues of $X + Y$ sorting.

REFERENCES

- [1] A. AHO, J. HOPCROFT AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] M. L. FREDMAN, *Two applications of a probabilistic search technique: Sorting $X + Y$ and building balanced search trees*, Proc. 7th Annual ACM Symp. on Theory of Computing (Albuquerque, NM), May, 1975, pp. 240-244.

- [3] F. GUSTAVSON AND D. Y. Y. YUN, *Arithmetic complexity of unordered sparse polynomials*, Proc. ACM Symp. on Symbolic and Algebraic Computation (Yorktown Heights, NY), Aug., 1976, pp. 149–153.
- [4] L. H. HARPER, T. H. PAYNE, J. E. SAVAGE AND E. STRAUS, *Sorting $X + Y$* , Comm. ACM, 18 (1975), pp. 347–349.
- [5] E. HOROWITZ, *A sorting algorithm for polynomial multiplication*, J. Assoc. Comput. Mach., 22 (1975), pp. 450–462.
- [6] S. C. JOHNSON, *Sparse polynomial arithmetic*, Proc. Eurosam Conf., SIGSAM Bull., 8 (1974), pp. 63–71.
- [7] D. A. KLIP, *Algebra by computer*, Technical Report for the Dept. of Computer and Information Sciences, University of Alabama, Birmingham, AL, September, 1973.
- [8] ———, *Some aspects of a portable algebra system*, Proc. ACM South-East Regional Conf. (Nashville, TN), April, 1974, pp. 1–22.
- [9] ———, *Different polynomial representations and their interaction in the portable algebra system PORT-ALG*, Proc. Eurosam Conf., SIGSAM Bull., 8 (1974), pp. 72–73.
- [10] ———, *The variable cell length listprocessor VARLIST*, Proc. ACM National Conf. (San Diego, CA), Nov. 1974, pp. 128–132.
- [11] ———, *The VARLIST listprocessing system*, Ref. Manual and Doc. Technical Report for the Dept. of Computer and Information Sciences, University of Alabama, Birmingham, AL, September, 1975.
- [12] D. E. KNUTH, *The Art of Computer Programming*, vol. 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1971.
- [13] R. MOENCK, *Practical fast polynomial multiplication*, Proc. ACM SYMSAC (Yorktown Heights, NY), August, 1976, pp. 136–148.
- [14] F. TEER, *Formula manipulation and PASCAL*, D.Sc. Thesis, Vrije Universiteit, Amsterdam, May, 1978.